Sprawozdanie

Operacje Macierzowe

Cel projektu:

Celem projektu jest pobranie macierzy z pliku csv, a następnie zaimplementowanie różnych matematycznych operacji na macierzach. Projekt został napisany w języku C, z wykorzystaniem odpowiednich struktur danych i algorytmów matematycznych. Projekt wykorzystuje tablice dynamiczne dwuwymiarowe, alokację oraz zwalnianie pamięci, struktury, a także obsługę błędów.

Działanie programu:

Po uruchomieniu programu w terminalu pojawia się uproszczony interfejs użytkownika. Wyświetlana jest numerowana lista operacji macierzowych oraz opcja zakończenia programu. Po dokonaniu wyboru użytkownik proszony jest o podanie nazwy pliku/plików .csv (ścieżki dostępu do pliku), a dla wykorzystania funkcji mnożenia macierzy przez skalar również jego wartości. Po dokonaniu operacji na macierzach użytkownik proszony jest o podanie nazwy pliku .csv do przechowania wyników działania. Dla funkcji obliczających wyznacznik macierzy oraz ślad wynik operacji wyświetlany jest w terminalu.
Program informuje na bieżąco o poprawności wczytania/zapisania plików oraz o ewentualnych błędach.

Elementy programu:

Plik *main.c*:

Funkcje *main* - wyświetla menu, a następnie przy pomocy konstrukcji switch case zagnieżdżonej w pętli while pozwala wybrać opcje działania. W przypadku wprowadzenia nieprawidłowego typu danej (innego niż integer) przerywa bieżące działanie pętli oraz oczyszcza buffor.
Funkcje *read_matrix_console* oraz *write_matrix_console* - dokonują odczytu/zapisu macierzy na podstawie wprowadzonej nazwy pliku (ścieżki dostępu).
Funkcje *nazwafunkcji_menu* - stanowią implementację odpowiednich funkcji operacji na macierzach dla poprawnego działania funkcji main. Sprawdzają również poprawność odczytu/zapisu plików oraz działania funkcji matematycznych. W tych funkcjach następuje zwolnienie wcześniej zaalokowanej pamięci dla macierzy (dynamicznych tablic dwuwymiarowych).

Plik *csv_operations.c*:

Funkcja *allocate_memory_for_matrix* - tworzy macierz dzięki dynamicznej alokacji pamięci (funkcja malloc()) na podstawie danych wymiarów macierzy. Dla niepoprawności w działaniu funkcji zwraca wcześniej zaalokowaną pamięć.

Funkcja *free_memory_for_matrix* - funkcja zwalniająca zaalokowaną pamięć dla macierzy.

Funkcja *read_matrix_from_csv* - odpowiedzialna za odczyt macierzy z pliku .csv i zwrócenie macierzy zawierającej odczytane dane. Sprawdza poprawność otwarcia pliku w trybie odczytu, w przypadku niepowodzenia zwraca macierz błędu o wartościach *NULL*. Następnie określa wymiary macierzy stosując konstrukcje przechodzenia między liniami pliku (dla liczby wierszy) oraz podziału linii dzięki zliczaniu tokenów między kolejnymi określonymi w pliku separatorami (dla liczby kolumn). Funkcja alokuje pamięć dla macierzy. Następnie ponownie przechodzi między kolejnymi wartościami w pliku w celu przypisania ich do zaalokowanej macierzy oraz dokonuje zamiany separatora dziesiętnego. Wyświetla informacje o prawidłowym odczycie.

Funkcja *write matrix_to_csv* - Sprawdza poprawność otwarcia pliku w trybie zapisu, w przypadku niepowodzenia zwraca *NULL*. Następnie wypisuje do pliku kolejne wartości z macierzy rozdzielające je separatorem, oraz znakiem końca linii dla kolejnego wiersza. W trakcie działania zmienia typ zmiennej na string, w celu ponownej zamiany separatora dziesiętnego. Wyświetla informacje o prawidłowym zapisie.

Plik *matrix_operations.c*:

Zawiera opracowane funkcje operacji na macierzach.

1. Dodawanie macierzy

   Algorytm polegający na dodaniu do siebie dwóch macierzy uzyskując jedną macierz wynikową. Działanie obsługuje funkcja **Matrix addition().**

   Opis kodu:

   1. Argumentami funkcji są macierze **matrixA** i **matrixB**.
   2. Sprawdzenie czy obie macierze mają te same wymiary i zwrócenie macierzy z błędem, w przypadku niespełnienia warunku.
   3. Alokacja pamięci dla macierzy wynikowej (**matrixR**).
   4. Główna pętla przechodzi po wszystkich wierszach.
   5. Pętla zagnieżdżona przechodzi po wszystkich kolumnach macierzy.
   6. Obliczanie wartości elementu macierzy wynikowej jako suma wartości z odpowiadającymi indeksami (**matrixR[i][j]=matrixA[i][j]+matrixB[i][j]**).
   7. Zwrócenie **matrixR**.

2. Odejmowanie macierzy

   Algorytm polegający na odjęciu do siebie dwóch macierzy uzyskując jedną macierz wynikową. Działanie obsługuje funkcja **Matrix subtraction().**

   Opis kodu:

   1. Argumentami funkcji są macierze **matrixA** i **matrixB**.
   2. Sprawdzenie czy obie macierze mają te same wymiary i zwrócenie macierzy z błędem, w przypadku niespełnienia warunku.
   3. Alokacja pamięci dla macierzy wynikowej (**matrixR**).
   4. Główna pętla przechodzi po wszystkich wierszach.

5. Pętla zagnieżdżona przechodzi po wszystkich kolumnach macierzy.
6. Obliczanie wartości elementu macierzy wynikowej jako różnica wartości z odpowiadającymi indeksami (**matrixR[i][j]=matrixA[i][j]-matrixB[i][j]**)
7. Zwrócenie **matrixR**.

3. Mnożenie macierzy przez skalar

Algorytm polegający na pomnożeniu macierzy przez liczbę i uzyskanie jednej macierzy wynikowej. Działanie obsługuje funkcja **Matrix multiplication_by_scalar().**

Opis kodu:

1. Argumentami funkcji jest macierz **matrixA** i liczba zmiennoprzecinkowa **scalar**.
2. Alokacja pamięci dla macierzy wynikowej (**matrixR**).
3. Główna pętla przechodzi po wszystkich wierszach.
4. Pętla zagnieżdżona przechodzi po wszystkich kolumnach macierzy.
5. Obliczanie elementu macierzy wynikowej jako wartości tego elementu macierzy pomnożone przez skalar (**matrixR[i][j] = matrixA[i][j] * scalar**).
6. Zwrócenie **matrixR**.

4. Mnożenie macierzy

Algorytm polegający na pomnożeniu macierzy przez drugą macierz i uzyskanie jednej macierzy wynikowej. Działanie obsługuje funkcja **Matrix multiplication().**

Opis kodu:

1. Argumentami funkcji są macierze **matrixA** i **matrixB**.
2. Sprawdzenie czy liczba kolumn w pierwszej macierzy jest taka sama jak liczba wierszy w drugiej macierzy i zwrócenie macierzy z błędem, w przypadku niespełnienia warunku.
3. Alokacja pamięci dla macierzy wynikowej (**matrixR**).
4. Główna pętla przechodzi po wszystkich wierszach.
5. Pętla zagnieżdżona przechodzi po wszystkich kolumnach macierzy.
6. Przypisanie wartości “0” do pojedynczego elementu z macierzy wynikowej.
7. Pętla zagnieżdżona o zmiennej “k” równemu warunkowi początkowemu przechodzi przez kolejne wartości dwóch macierzy w podanym wierszu i kolumnie i oblicza wynik jako suma iloczynów odpowiadających elementów.
8. Zwrócenie **matrixR**.

5. Transpozycja

Algorytm polegający na transpozycji macierzy i uzyskanie jednej macierzy wynikowej. Działanie obsługuje funkcja **Matrix transposition().**

Opis kodu:

1. Argumentami funkcji jest macierz **matrixA**.
2. Alokacja pamięci dla macierzy wynikowej (**matrixR**).
3. Główna pętla przechodzi po wszystkich wierszach.
4. Pętla zagnieżdżona przechodzi po wszystkich kolumnach macierzy.
5. Obliczanie elementu macierzy wynikowej poprzez zamianę indeksów wierszy i kolumn (**matrixR[j][i] = matrixA[i][j]**)
6. Zwrócenie **matrixR**.

6. Ślad macierzy

Algorytm polegający na zsumowaniu wartości elementów macierzy leżących na głównej przekątnej. Działanie obsługuje funkcja **trace().**

Opis kodu:

1. Argumentem funkcji jest macierz **matrixA**.
2. Sprawdzenie czy liczba kolumn i wierszy jest taka sama zwrócenie wartości *NAN*, w przypadku niespełnienia warunku.
3. Iteracja po wierszach macierzy.
4. Obliczenie śladu, jako zsumowanie wartości elementów macierzy o liczbie wierszy równej liczbie kolumn (**matrixA[i][i]**).
5. Zwrócenie **matrixR**.

7. Rozkład LU

Algorytm z jednej macierzy kwadratowej uzyskuje dwie macierze trójkątne zgodne z rozkładem LU. Działanie obsługuje funkcja **Matrices LU_decomposition().**

Opis kodu:

1. Argumentem funkcji jest macierz **matrixA** .
2. Sprawdzenie czy macierz jest kwadratowa i zwrócenie macierzy z błędem w przypadku niespełnienia warunku.
3. Alokacja pamięci dla macierzy U i L.
4. Pętla zagnieżdżona ustawiająca wartości w macierzach U i L na ‘0’.
5. Pętla główna przechodzi przez wszystkie wiersze macierzy.
6. Wartości na przekątnej macierzy L są ustawiane na 1 zgodnie z definicją macierzy dolnotrójkątnej L.
7. Obliczenie wartości macierzy U: Dla każdej kolumny j od i do n, przypisujemy wartość z oryginalnej macierzy do U[i][j].
   Następnie korygujemy wartość U[i][j] odejmując sumę iloczynów odpowiednich elementów z macierzy L i U, które zostały wcześniej obliczone. To usuwa wpływ wcześniejszych wierszy, aby zachować właściwości macierzy trójkątnej.
8. Obliczenie wartości macierzy L: Dla każdej kolumny j od i+1 do n, przypisujemy wartość z oryginalnej macierzy do L[j][i].

Następnie korygujemy wartość L[j][i] w ten sam sposób co w punkcie **7.** dla macierzy U.
Na końcu, dzielimy L[j][i] przez wartość na przekątnej U[i][i].

9. Wyjście z pętli i utworzenie struktury wynikowej **Matrices** oraz spakowanie do niej dwóch macierzy wynikowych.
10. Zwrócenie **R** jak struktury {**L**, **U**}**.**

8. Wyznacznik macierzy

Algorytm z jednej macierzy kwadratowej uzyskuje wyznacznik tej macierzy. Działanie obsługuje funkcja **float determinant_with_LU().** Wyznacznik jest obliczany wydajną metodą z wykorzystaniem rozkładu LU (złożoność obliczeniowa n^3). Algorytm wydajnie obsługuje macierze również o dużych rozmiarach. Sam wyznacznik jest obliczany poprzez przemnożenie przez siebie wartości znajdujących się na przekątnej macierzy trójkątnej U z rozkładu LU.

Opis kodu:

1. Argumentem funkcji jest macierz **matrix**.
2. Wywołanie funkcji **LU_decomposition()** i pobranie dwóch macierzy wynikowych do zmiennej **result.**
3. Sprawdzenie czy macierze L i U zostały utworzone poprawnie.
4. Jeśli Tak, zgodnie z matematycznym algorytmem funkcja przemnaża wartości znajdujące się na przekątnej macierzy U (matrix2) uzyskując wartość wyznacznika. Następnie zwrócenie wyznacznika **det**.
5. Jeśli Nie, zwrócenie wartości *NAN*.

9. Macierz dopełnień

Algorytm uzyskuje z macierzy kwadratowej jedną macierz dopełnień, która będzie w następnym kroku wykorzystana do stworzenia macierzy odwrotnej. Działania obsługują funkcje **cofactor_matrix()** oraz **create_minor().**

Opis kodu:

1. Argumentem funkcji **cofactor_matrix()** jest macierz **matrix.**
2. Sprawdzenie czy macierz wejściowa jest kwadratowa, jeśli nie zwrócenie macierzy błędów.
3. Alokacja pamięci dla macierzy wynikowej.
4. Przechodzenie po każdej wartości macierzy wejścia.
5. Z każdą iteracją uruchomienie funkcji **create_minor().**
6. Funkcja **create_minor()** pobiera macierz oraz numer kolumny **j** i wiersza **i**.
7. Alokuje pamięć dla macierzy o 1 mniejszej.
8. Zgodnie z matematyczną definicją pomija wiersz **i** i kolumnę **j.** Resztę wartości przekleja do minora.
9. Funkcja zwraca **minor**.

10. Już w funkcji **cofactor_matrix()** zgodnie z matematyczną definicją wyliczana jest wartość komórki **[i,j]** macierzy dopełnień: (-1)^(i+j)*det(minor). Następuje wykorzystanie funkcji obliczającej wyznacznik
11. Po wszystkich iteracjach następuje zwrócenie macierzy dopełnień - **cofactor.**

10. Odwracanie macierzy

Algorytm uzyskuje z macierzy kwadratowej jedną macierz odwrotną. Działanie obsługuje funkcja **inverse_matrix().** Funkcja wykorzystuje poprzednio opisane funkcje.

Opis kodu:

1. Argumentem funkcji **inverse_matrix()** jest macierz **matrix**.
2. Obliczenie wyznacznika macierzy przy pomocy funkcji **determinant_with_LU()** i przypisanie wyniku do zmiennej **det**.
3. Sprawdzenie czy macierz jest odwracalna (det != 0), jeśli nie zwrócenie macierzy błędów
4. Obliczenie macierzy dopełnień przy pomocy funkcji **cofactor_matrix()** i przypisanie wyniku do zmiennej **cofactor**.
5. Transpozycja macierzy **cofactor** za pomocą funkcji **transposition()** i przypisanie wyniku do zmiennej **adjugate**.
6. Zwolnienie pamięci macierzy **cofactor** i alokacja pamięci wynikowej odwrotnej.
7. Zgodnie z matematyczną definicją obliczenie elementów macierzy odwrotnej **inverse**. Każda wartość to odpowiadająca wartość macierzy dopełnień po transpozycji podzielona przez wyznacznik macierzy.
8. Zwolnienie pamięci macierzy **adjugate** i zwrócenie macierzy wynikowej **inverse**.

Inne pliki:

csv_operations.h oraz *matrix_operations.h* - pliki nagłówkowe utworzone dla prawidłowego działania powyższych funkcji.

Zastosowane struktury:

W projekcie zostały wykorzystane dwie struktury, które przechowywane są w pliku nagłówkowym *csv_operations.h*.

Struktura *Matrix* przechowuje informacje o macierzach:

- rows - liczba wierszy macierzy
- columns - liczba kolumn macierzy
- values - przechowująca wartości elementów macierzy.

Struktura *Matrices* przechowuje informacje o dwóch macierzach wynikowych:

- matrix1 - pierwsza struktura *Matrix*
- matrix2 - druga struktura *Matrix*

Inne wymagania:

Program przeznaczony jest do operacji na plikach .csv o kodowaniu UTF-8, rozdzielonych przecinkami.
Dla celów testowych do ogólnego programu zostały dodane cztery pliki spełniające wymogi.

Podział pracy:

Wiktoria Wróbel:

- nadzór projektu
- opracowanie algorytmów
    - odejmowania macierzy
    - dodawanie macierzy
    - mnożenia macierzy
    - transponowania macierzy
    - wyznaczania śladu macierzy
- sprawdzenie poprawności działania opracowanych algorytmów oraz odporności programu na błędy
- przygotowanie sprawozdania

Wiktor Suchoński-Romaniuk:

- opracowanie zastosowanych struktur
- opracowanie algorytmów:
    - mnożenie macierzy przez skalar
    - dekompozycja LU
    - obliczanie wyznacznika macierzy
    - macierz dopełnień
    - macierz odwrotna
- przygotowanie sprawozdania

Sebastian Szklarski:

- opracowanie funkcji związanych z przetwarzaniem plików. csv oraz alokacją pamięci
- utworzenie uproszczonego interfejsu użytkownika
- implementacja funkcji macierzowych na potrzeby działania menu
- przygotowanie sprawozdania